Dual-CBS: A Hierarchical Approach via Conflict-Based Search and Sampling for Multi-Agent Motion Planning

Joonyeol Sim[*1], **Shao-Hung Chan**[*2], **Sven Koenig**[1,3]
[1]University of California, Irvine
[2]University of Southern California
[3]University of Örebro
joonyeos@uci.edu, shaohung@usc.edu, svenk@uci.edu

Abstract

Multi-Agent Motion Planning (MAMP) is the problem of finding a set of collision-free trajectories in a configuration space, one for each agent, to move from their start configurations to their goal configurations while minimizing their sum of travel times. To solve MAMP, one approach is to construct a roadmap offline from the entire configuration space and search for a solution accordingly. However, constructing a roadmap and resolving collisions between agents on the roadmap can result in heavy computational overhead. Another approach is to construct a roadmap for each agent online. It assigns priorities between agents and samples one trajectory for each agent on its roadmap, where agents with low priorities should avoid collisions from those with high priorities. However, agents with low priorities may take a long time to reach their goal configurations, and thus the solution this approach finds may have a high sum of travel times. To combine search and sampling, we propose *Dual Conflict-Based Search* (Dual-CBS). Its strategy is to iteratively find one trajectory for each agent and resolve collisions between them. To find a trajectory for each agent, Dual-CBS constructs a four-neighbor grid graph, where each grid cell represents a square area of the configuration space. Then, it runs a search algorithm to find one grid-based path for each agent on the graph, and samples one configuration in each grid cell. To resolve collisions, Dual-CBS uses *swept area constraints* to prevent an agent from traversing areas that overlap with the trajectories of other agents. Compared to the approach that constructs roadmaps offline, Dual-CBS incurs significantly less computational overhead in constructing the roadmap. Compared to the approach that constructs roadmaps online, Dual-CBS guides the sampled trajectories with the grid-based paths and thus finds solutions with a lower sum of travel times.

Introduction

Multi-Agent Motion Planning (MAMP) is the problem of finding a set of collision-free trajectories, one for each agent, to move from their start locations to their goal locations in a continuous environment while minimizing the sum of travel time (Okumura and Défago 2023; Cohen et al. 2019). It has numerous applications, including operating a swarm of drones, navigating a group of autonomous vehicles, and coordinating a team of mobile robots in an automated warehouse. However, MAMP is a challenging problem. Compared to Single-Agent Motion Planning (SAMP), which aims to find a path that avoids collisions with static obstacles, MAMP requires multiple paths to avoid collisions with static obstacles and other agents, making it intractable as the number of agents increases. Thus, recent studies solve MAMP with a two-phase approach that (I) avoids collisions with static obstacles by constructing a roadmap and (II) avoids collisions between agents by running a search algorithm on the roadmap (Okumura and Défago 2023). A *roadmap* is a graph with vertices representing the configurations where an agent does not collide with any static obstacles and with edges representing the transitions (i.e., actions) between two configurations.

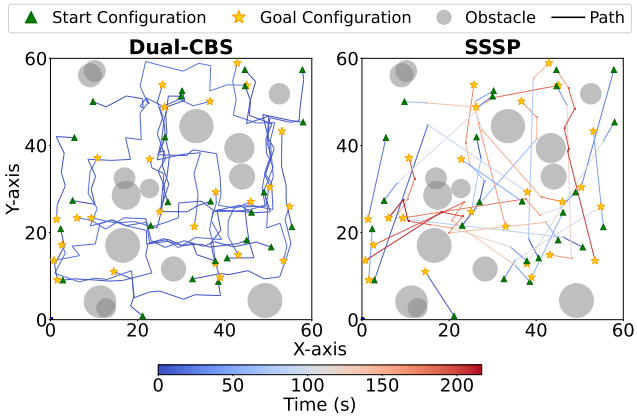


Figure 1: The trajectory visualizations of our Dual-CBS and the state-of-the-art MAMP approach, SSSP. The colors of a trajectory, ranging from blue to red, indicate its travel time (in seconds), progressing from low to high. Although SSSP generates trajectories of smaller lengths than Dual-CBS, it requires agents to move sequentially to avoid collisions and thus introduces larger delays than Dual-CBS. In this MAMP instance, Dual-CBS and SSSP find solutions with a sum of travel time of 26.63 seconds and 216.60 seconds. Dual-CBS achieves approximately 8 times lower sum of travel time than SSSP.

[*]These authors contributed equally.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

One approach is to construct a roadmap *offline* (or, equivalently, explicitly) as a pre-processing technique that covers the configuration space (Henkel and Toussaint 2020; Hönig et al. 2018). Although offline approaches typically generate trajectories of small travel times, they mainly suffer from two drawbacks. Firstly, if the configuration space is large, then constructing a roadmap can result in a huge computational overhead. Secondly, if the roadmap is sparse, then it can limit the transitions of agents' configurations, resulting in difficulties in resolving collisions. In contrast, one can also construct a roadmap *online* (or, equivalently, implicitly) that incrementally generates its vertices and edges (Okumura and Défago 2023; Kottinger, Almagor, and Lahijanian 2022). Although online approaches can find a set of collision-free paths in a short runtime, they generate trajectories with large travel times.

To find trajectories that avoid collisions (i.e., in Phase (II)), one approach is to run a search algorithm by viewing the configurations of all agents as a *joint state*. However, searching the joint-state space to find a near-optimal solution can result in a large branching factor, leading to inefficiency. Thus, recent studies have focused on decoupling MAMP into several prioritized SAMP problems, where low-priority agents should avoid collisions with high-priority agents. On the other hand, since solving even a simple form of an SAMP problem is PSPACE-hard (Reif 1979), sampling-based search algorithms are employed to expedite the process (Karaman et al. 2011). However, as the configuration space increases, sampling-based search algorithms can result in huge computational overhead and, meanwhile, produce trajectories with large travel times.

In this paper, to strike a balance between runtime and solution quality, we propose *Dual Conflict-Based Search* (Dual-CBS), which combines search and sampling in a hierarchical manner. Dual-CBS has two components: the *Planning CBS* (PCBS) and the *Sampling CBS* (SCBS). PCBS discretizes time into timesteps and the configuration space into a four-neighbor grid graph, where each grid cell represents a square area that can be occupied by multiple agents. Dual-CBS uses PCBS to *search* for a set of shortest grid-based paths and reduce the number of *conflicts*, which happen when two agents visit the same grid cell at the same timestep or traverse the same edge in opposite directions at the same timestep. Then, given a set of grid-based paths from PCBS (which can still contain conflicts), Dual-CBS uses SCBS to *sample* a set of trajectories, where each grid cell contains a sampled configuration. Timesteps are mapped to time intervals. To resolve collisions between agents when sampling trajectories, we propose *swept volume constraints* to prevent agents from overlapping with each other's trajectories inside the same grid at the same time interval.

Dual-CBS finds a set of grid-based paths and then samples the trajectories during the search accordingly. Thus, it can be viewed as constructing a customized roadmap for each agent along its grid-based path, rather than constructing a roadmap for the entire configuration space. That is, the set of grid-based paths from PCBS reduces the sampling space of SCBS to sequences of grid cells. Additionally, since PCBS always finds the shortest grid-based paths, its grid-based paths provide guidance for SCBS to sample trajectories that are near-optimal, resulting in solutions with a low sum of travel times.

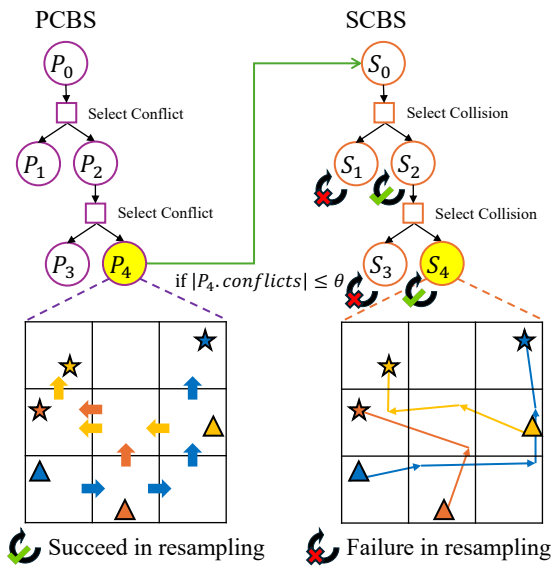
Related Work

Single-Agent Motion Planning (SAMP) A typical approach to solving an SAMP problem uses a search algorithm to find a path that does not collide with static obstacles. One approach is to construct a roadmap offline in the configuration space and then use a search algorithm to find a path on it. In contrast, another approach is to avoid collisions with static obstacles during the search. Sampling-based search algorithms have shown their efficiency in solving SAMP. For example, the Probabilistic Roadmap (PRM) (Kavraki et al. 1996) is a sampling-based roadmap, while the Rapidly Exploring Random Tree (RRT) (LaValle 1998) is a search tree that explores the configuration space by sampling and performs collision checking during the search.

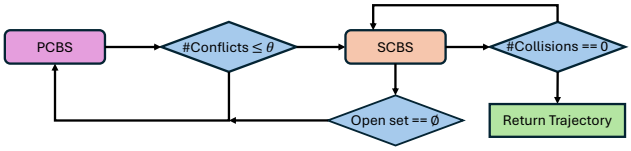
Multi-Agent Motion Planning (MAMP) We focus on the aforementioned two-phase approaches. In terms of constructing roadmaps offline, Henkel and Toussaint (2020) construct the roadmap by first generating samples with PRM in the configuration space and then optimizing the positions of the resulting vertices and edges via Stochastic Gradient Descent. Hönig et al. (2018) construct the roadmap with SPArse Roadmap Spanner algorithm (SPARS) (Dobson and Bekris 2014). Although offline construction of the roadmap typically yields near-optimal solutions for MAMP, it introduces huge computational overhead for roadmap construction. On the other hand, in terms of constructing roadmaps online, Okumura and Défago (2023) develop *Simultaneous Sampling-and-Search Planning* (SSSP), which simultaneously generates a roadmap for each agent and runs a search algorithm to find a collision-free trajectory on each roadmap. Although SSSP finds a set of collision-free trajectories fast, it resolves collisions between agents with priorities, where agents with low priorities should avoid collisions with those with high priorities. In this case, agents with low priorities may require long travel times to reach their goal configurations. Thus, SSSP tends to find solutions with large sums of travel times, as shown in Figure 1.

Problem Definition

We follow the definition of Multi-Agent Motion Planning (MAMP) as presented in Okumura and Défago (2023). We consider a MAMP problem in a workspace $\mathcal{W} \subseteq \mathbb{R}^2$ that represents the physical environment in which agents move. The agent set $\mathcal{A} = \{a_1, ..., a_n\}$ consists of n agents. All agents operate in a shared configuration space $\mathcal{C} \subseteq \mathbb{R}^2$. The configuration of agent a_i is denoted as $q_i \in \mathcal{C}$. The workspace occupied by an agent at configuration q is denoted as $\mathcal{R}(q) \subset \mathcal{W}$. For obstacle regions $\mathcal{O} \subset \mathcal{W}$, the free configuration space is defined as $\mathcal{C}^{\text{free}} := \{q \in \mathcal{C} \mid \mathcal{R}(q) \cap \mathcal{O} = \emptyset\}$. A trajectory of agent a_i is defined as $\sigma_i : \mathbb{R}_{\geq 0} \to \mathcal{C}^{\text{free}}$, mapping continuous times to configurations in the free configuration space. We denote the set of all trajectories as Σ. Each agent $a_i \in \mathcal{A}$ has a start configuration $s_i \in \mathcal{C}^{\text{free}}$ and a goal configuration $g_i \in \mathcal{C}^{\text{free}}$. A



(a) PCBS and SCBS tree expansion



(b) Dual-CBS flowchart

Figure 2: Overview of Dual-CBS. A PCBS node has a set of grid-based paths, and an SCBS node contains trajectories. (a) PCBS expands PCBS nodes until the number of conflicts drops below the threshold θ at node P_4, then passes a set of grid-based paths stored in P_4 to SCBS for continuous trajectory adjustment. The stars and triangles represent the goal and start configurations of the agent. (b) The control flow between PCBS and SCBS. If an SCBS node has no collisions, the algorithm returns the collision-free trajectories. If the open set of SCBS becomes empty, SCBS transfers control to PCBS to find an alternative set of grid-based paths.

collision $\langle a_i, a_j, t_{\text{start}}, t_{\text{end}} \rangle$ occurs between agents a_i and a_j that follows trajectories σ_i and σ_j, when their bodies overlap in the workspace, i.e., $\mathcal{R}(\sigma_i(t)) \cap \mathcal{R}(\sigma_j(t)) \neq \emptyset$, for some $t \in [t_{\text{start}}, t_{\text{end}}]$. The travel time T_i of agent a_i is the time at which it arrives at its goal configuration g_i and remains there. The completion time $T_{\text{end}} = \max_i T_i$ is the time at which all agents have reached their goal configurations.

We aim to find collision-free trajectories Σ that minimize the sum of travel times $\sum_i T_i$, where each trajectory σ_i satisfies:

- $\sigma_i(0) = s_i \wedge \sigma_i(T_{\text{end}}) = g_i$
- $\sigma_i(t) \in \mathcal{C}^{\text{free}}, \forall t \in [0, T_{\text{end}}]$
- $\mathcal{R}(\sigma_i(t)) \cap \mathcal{R}(\sigma_j(t)) = \emptyset, \forall j \neq i, t \in [0, T_{\text{end}}]$

Dual Conflict-Based Search (Dual-CBS)

Dual Conflict-Based Search (Dual-CBS) alternates between a discrete MAPF solver and a continuous trajectory adjustment. The discrete MAPF solver finds a set of grid-based paths that constrain a continuous sampling space. The continuous trajectory adjustment then generates collision-free trajectories by iteratively sampling configurations within the grid cells traversed by these paths.

Dual-CBS comprises two components: Planning CBS (PCBS) for the discrete MAPF solver and Sampling CBS (SCBS) for the continuous trajectory adjustment. Before running Dual-CBS, we discretize the workspace into a four-neighbor grid graph by partitioning the environment into grid cells of length L. PCBS then operates on this grid graph and expands the PCBS tree until it finds a node whose number of conflicts drops below a given conflict threshold θ. Once this condition is met, PCBS passes the current set of grid-based paths stored in that node to SCBS. SCBS then operates on the configuration space $\mathcal{C}$ and performs continuous trajectory adjustment by expanding the SCBS tree until all trajectories are collision-free. If SCBS fails to resolve all collisions, it discards the current SCBS tree and transfers control to PCBS to obtain an alternative set of grid-based paths. This alternation between PCBS and SCBS continues until a collision-free solution is found. Figure 2 illustrates this alternating process between PCBS and SCBS.

Dual-CBS has a few parameters. The sampling attempt limit N_{att} is the maximum number of sampling attempts allowed at each SCBS node to resolve a collision. The conflict threshold θ determines when PCBS transfers control to SCBS. Once the number of conflicts in a PCBS node drops below θ, PCBS passes a set of grid-based paths to SCBS. The maximum velocity $v_{\max}$ limits the speed of all agents to ensure kinematic feasibility.

Planning Conflict-Based Search (PCBS)

The four-neighbor grid graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the vertex set and $\mathcal{E}$ represents the edge set. A path of agent a_i is defined as $\pi_i : \mathbb{Z}_{\geq 0} \to \mathcal{V}$, mapping discrete timesteps to vertices in the grid graph. We denote the set of all grid-based paths as Π. We consider three types of conflicts in the four-neighbor grid graph $\mathcal{G}$. A *vertex conflict* $\langle a_i, a_j, v, t \rangle$ occurs when agents a_i and a_j occupy the same vertex $v \in \mathcal{V}$ at the same timestep t. A vertex conflict is a *target conflict* if one agent has arrived at its goal grid cell and remains there permanently while the other agent passes through the same vertex. An *edge conflict* $\langle a_i, a_j, u, v, t \rangle$ occurs when agents a_i and a_j traverse the same edge $(u, v) \in \mathcal{E}$ in opposite directions from timestep $t-1$ to timestep t.

We define four constraint types: (i) a *vertex constraints* forbid agent a_i from occupying vertex v at timestep t, (ii) an *edge constraints* forbid agent a_i from traversing edge (u, v) from timestep $t-1$ to timestep t, (iii) *length constraints* requires agent a_i to delay its arrival at vertex v until at least timestep $t+1$, and (iv) *target-block constraints* forbid agent a_i from traversing vertex v at or after timestep t.

PCBS follows the two-level framework of Conflict-Based Search (CBS) (Sharon et al. 2015), which separates the

Algorithm 1: Planning CBS (PCBS)

```
1: procedure DUAL-CBS(A, N_att, θ, v_max)
2:     P_root ← new PCBSNode
3:     P_root.Π ← GENERATEINITIALPATHS(A)
4:     P_root.cost ← COMPUTECOST(P_root.Π)
5:     P_root.conflicts ← DETECTCONFLICTS(P_root.Π)
6:     P_root.constraints ← ∅
7:     Q_PCBS ← {P_root}
8:     while Q_PCBS ≠ ∅ do
9:         P ← Q_PCBS.pop()
10:        if |P.conflicts| ≤ θ then
11:            Σ ← SOLVESCBS(P.Π, N_att, v_max)
12:            if Σ ≠ None then
13:                return Σ
14:        conflict ← CHOOSECONFLICT(P.conflicts)
15:        for each agent a_i in conflict do
16:            P' ← COPY(P)
17:            P'.constraints ← P'.constraints
                   ∪ {GENERATECONSTRAINT(conflict, a_i)}
18:            π_i ← FINDPATH(a_i, P'.constraints)
19:            if π_i ≠ None then
20:                P'.Π[i] ← π_i
21:                P'.cost ← COMPUTECOST(P'.Π)
22:                P'.conflicts ← DETECTCONFLICTS(P'.Π)
23:                Q_PCBS ← Q_PCBS ∪ {P'}
24:    return No solution found
```

Algorithm 2: Sampling CBS (SCBS)

```
1: procedure SOLVESCBS(Π, N_att, v_max)
2:     S_root ← new SCBSNode
3:     S_root.Σ ← SAMPLEINITIALTRAJECTORIES(Π)
4:     S_root.cost ← COMPUTECOST(S_root.Σ)
5:     S_root.collisions ← DETECTCOLLISIONS(S_root.Σ)
6:     S_root.constraints ← ∅
7:     Q_SCBS ← {S_root}
8:     while Q_SCBS ≠ ∅ do
9:         S ← Q_SCBS.pop()
10:        if |S.collisions| = 0 then return S.Σ
11:        collision ← CHOOSECOLLISION(S.collisions)
12:        t ← FINDASSOCIATEDTIMESTEP(S, collision)
13:        for each agent a_i in collision do
14:            S' ← COPY(S)
15:            S'.constraints ← S'.constraints
                   ∪ {GENERATECONSTRAINT(collision, a_i, t)}
16:            for att ← 1 to N_att do
17:                q' ← SAMPLING(S'.constraints, t)
18:                if q' ≠ None then
19:                    S'.Σ[i][t] ← q'
20:                    if CHECKCONSTRAINTS(S', a_i, t) then
21:                        S'.cost ← COMPUTECOST(S'.Σ)
22:                        S'.collisions ←
                               DETECTCOLLISIONS(S'.Σ)
23:                        Q_SCBS ← Q_SCBS ∪ {S'}
24:                        break
25:    return None
```

search into a high-level constraint tree expansion and low-level single-agent path planning. At the high level, CBS expands a constraint tree where each node contains a set of constraints imposed on each agent. CBS expands nodes in a best-first manner according to the sum of path lengths. At the low level, CBS uses A* to replan paths of individual agents subject to the constraints of the current node. When a conflict is detected in the current node, CBS splits that node into two child nodes. Each child node adds a corresponding constraint to one of the conflicting agents.

Algorithm 1 presents PCBS. PCBS maintains the high-level and low-level search structure of CBS. It begins by initializing the root node $\mathcal{P}_{\text{root}}$ (Line 2). It then generates initial paths for all agents using A* and computes the sum of path lengths (Lines 3–4). PCBS detects conflicts and initializes the constraint set to empty (Lines 5–6). This root node is then inserted into a priority queue Q_{PCBS} (Line 7). At each iteration, PCBS expands the node $\mathcal{P}$ with the lowest sum of path lengths from Q_{PCBS} and removes it from Q_{PCBS} (Line 9). Unlike CBS, PCBS first checks whether the number of conflicts of node $\mathcal{P}$ drops below the conflict threshold θ (Line 10). If so, then the set of paths of that node $\mathcal{P}.\Pi$ is passed to SCBS (Line 11). If SCBS successfully resolves all collisions and returns collision-free trajectories, then PCBS returns those trajectories and terminates (Lines 12–13). Otherwise, PCBS selects the first conflict from $\mathcal{P}$.conflicts (Line 14). For each agent involved in the conflict, PCBS creates a child node by copying the parent node (Lines 15–16).

For a vertex conflict, PCBS splits $\mathcal{P}$ into two child nodes by imposing a vertex constraint on one of the conflicting agents. Similarly, for an edge conflict, PCBS imposes an edge constraint. For a target conflict, where one agent a_i occupies its goal grid cell while some other agents pass through at timestep t, PCBS splits $\mathcal{P}$ into two child nodes. One with a length constraint on a_i that delays the arrival of agent a_i at its goal grid cell by at least one timestep (i.e., the time agent a_i is allowed to stay permanently at its goal grid cell should be at least $t + 1$), and another one with target-block constraints on all agents that pass through the goal grid cell of agent a_i. These constraints prevent them from traversing it at or after timestep t (Line 17).

The path of the constrained agent is then replanned to respect the imposed constraint and the constraints imposed on it by the parent nodes of $\mathcal{P}$ (Line 18). If a path is found, the solution of the child node, its sum of path lengths, and conflicts are updated (Lines 20–22), and the node is inserted into Q_{PCBS} (Line 23). This best-first search continues until either SCBS finds collision-free trajectories or Q_{PCBS} becomes empty, in which case the algorithm returns no solution (Line 24).

Sampling Conflict-Based Search (SCBS)

SCBS takes a set of grid-based paths from PCBS and generates trajectories by sampling one configuration within each grid cell traversed by the grid-based paths. It then expands the SCBS tree to find collision-free trajectories.

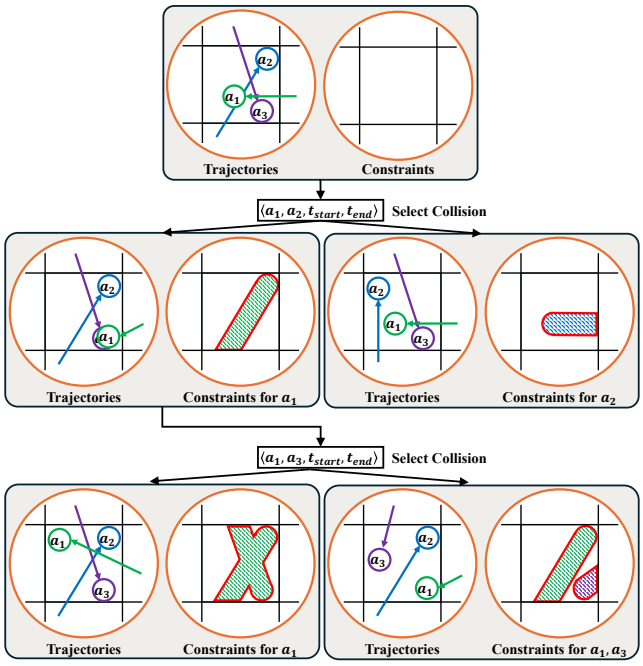


Figure 3: Illustration of the SCBS tree with swept area constraints. SCBS detects a collision $\langle a_1, a_2, t_{\text{start}}, t_{\text{end}} \rangle$ in the root node (inside the box) and splits the root node into two child nodes, each adding a swept area constraint for one of the colliding agents. The color of each swept area corresponds to the agent on which the constraint is imposed. In the left child node, the configuration for a_1 is sampled to avoid the swept area of a_2, and vice versa in the right child node. SCBS then chooses the left child node, detects a collision $\langle a_1, a_3, t_{\text{start}}, t_{\text{end}} \rangle$, and splits it into two child nodes. Each node maintains constraints inherited from its ancestor nodes and a newly added constraint. In the bottom-left node, a_1 must avoid sampling within the swept areas of both a_2 and a_3.

Agent Synchronization While PCBS operates on discrete timesteps, SCBS maps each timestep to a continuous time interval. SCBS determines the velocity of each agent using sampled configurations so that they arrive at their next configurations simultaneously. Along the set of grid-based paths Π, each agent a_i moves from q_i^t to q_i^{t+1} from timestep t to timestep $t+1$. To ensure all agents arrive at their next configurations at the same time, the timestep duration $\Delta\tau_t$ is set to the time required for the agent with the largest displacement to travel at maximum velocity $v_{\max}$. The velocity of each agent a_i at timestep t is then computed as:

$$\Delta\tau_t = \frac{\max_j \|q_j^{t+1} - q_j^t\|_2}{v_{\max}}, \quad v_i^t = \frac{q_i^{t+1} - q_i^t}{\Delta\tau_t}. \quad (1)$$

where $q_i^{t+1} - q_i^t$ is the displacement vector and $\|\cdot\|_2$ denotes the Euclidean norm in $\mathbb{R}^2$. By construction, the agent with the largest displacement in timestep t travels exactly at speed $v_{\max}$, while every other agent travels no faster than $v_{\max}$ to ensure their arrivals at the end of the timestep.

Swept Area Constraints SCBS introduces *swept area constraints*, which are inspired by the concept of swept volumes (Abdel-Malek et al. 2006). A swept volume represents the total space traced by a moving object over time. We adapt this concept to our 2D workspace as the *swept area*, defined as the spatial region occupied by the trajectory of an agent during a given time interval. When a collision is detected, SCBS samples another configuration for one of the colliding agents to avoid that collision. Since the sampled configuration determines where the agent arrives, sampling within the swept area of the trajectory of the other agent involved in the collision causes another collision. A swept area constraint is defined as $\langle a_i, \sigma_j, t_{\text{start}}, t_{\text{end}} \rangle$, which restricts SCBS from sampling configurations for agent a_i that enter the swept area of trajectory σ_j during the time interval $[t_{\text{start}}, t_{\text{end}}]$, i.e., $\mathcal{R}_i(\sigma_i(t)) \cap \left(\bigcup_{\tau \in [t_{\text{start}}, t_{\text{end}}]} \mathcal{R}_j(\sigma_j(\tau)) \right) = \emptyset$ for all $t \in [t_{\text{start}}, t_{\text{end}}]$.

Algorithm 2 presents SCBS. It begins by initializing the root node (Line 2). For each agent a_i, SCBS generates an initial trajectory by sampling one configuration within each grid cell traversed by its grid-based path π_i. The start configuration s_i and goal configuration g_i remain fixed, while configurations for intermediate grid cells are sampled from $\mathcal{C}^{\text{free}}$ within each cell (Line 3). Using the velocities computed from Equation 1, SCBS calculates the sum of travel times and detects collisions (Lines 4–5). The constraint set is initialized to empty (Line 6). The root node is then inserted into a priority queue Q_{SCBS} ordered by sum of travel times (Line 7).

At each iteration, SCBS expands the node $\mathcal{S}$ with the lowest sum of travel times from Q_{SCBS} and removes it from Q_{SCBS} (Line 9). If no collisions remain, SCBS returns the collision-free trajectories (Line 10). Otherwise, SCBS selects the first collision (Line 11). Since collisions occur in continuous time, SCBS maps the collision to the corresponding discrete timestep t along the grid-based path. Due to agent synchronization, all agents transition from timestep $t-1$ to t during the same time interval. Thus, if a collision occurs within that interval, SCBS identifies t as the associated timestep (Line 12). SCBS then splits the node into two child nodes (Line 13). For each conflicting agent a_i, SCBS creates a child node $\mathcal{S}'$ by copying $\mathcal{S}$ (Line 14) and adding a swept area constraint $\langle a_i, \sigma_j, t_{\text{start}}, t_{\text{end}} \rangle$, where $[t_{\text{start}}, t_{\text{end}}]$ corresponds to the time interval between timesteps $t-1$ and t (Line 15). Figure 3 illustrates this process.

SCBS then attempts to sample the configuration of agent a_i at timestep t up to N_{att} times (Line 16). At each attempt, SCBS samples a new configuration q' in the grid cell at timestep t along the path of agent a_i (Line 17). If such a configuration that satisfies the swept area constraints is found (Line 18), the configuration in the trajectory of agent a_i at timestep t is updated (Line 19). SCBS then checks whether agent a_i collides with any agent specified in its constraints during the transition from timestep $t-1$ to t (Line 20). If the constraints are satisfied, the cost and collisions of the child node are updated (Lines 21–22). The child node is then inserted into Q_{SCBS} (Line 23), and the sampling loop terminates (Line 24).

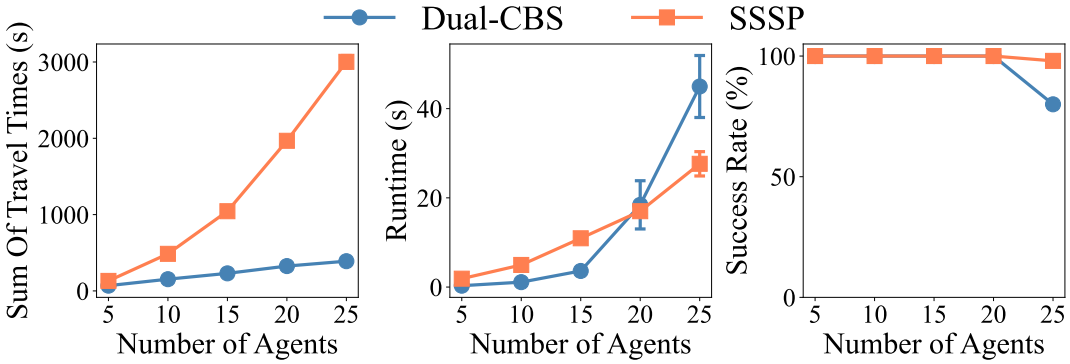


Figure 4: Performance comparison between Dual-CBS and SSSP for different numbers of agents, showing the sum of travel times, runtime, and success rate.

The best-first node expansions continue until either collision-free trajectories are found or Q_{SCBS} becomes empty (Line 25). In the latter case, SCBS returns None and transfers control to PCBS to find an alternative set of grid-based paths.

Experiments

We compare Dual-CBS to SSSP (Okumura and Défago 2023), one of the state-of-the-art online approaches for the MAMP problem. We generate test environments with circular obstacles of varying radii. The environment size is set to $60\,\text{m} \times 60\,\text{m}$, and all agents have a radius of $0.5\,\text{m}$. The obstacles are randomly placed until the obstacle density reaches 10%. We allow the obstacles to overlap, creating larger non-convex obstacles. We generate 50 problem instances for each number of agents in $\{5, 10, 15, 20, 25\}$, with random obstacle locations and radii, and start and goal configurations of all agents sampled randomly. An example environment is shown in Figure 1.

The maximum velocity v_{max} is set to $4.0\,\text{m/s}$ for all agents. As described in the agent synchronization, Dual-CBS adjusts the velocity of each agent so that all agents arrive at their next configurations simultaneously. In contrast, SSSP plans sequentially for each agent, allowing each to move at maximum velocity without synchronization constraints. The parameter values for Dual-CBS are set to $v_{\text{max}} = 4.0$, $N_{\text{att}} = 20$, and $\theta = 20$. We set a time limit of 300 seconds for each problem instance. All experiments are conducted on an Intel Xeon Gold 6148 2.40 GHz CPU and 192 GB RAM. Both Dual-CBS and SSSP are implemented in Python 3.12.

We evaluate the performance of Dual-CBS and SSSP using four metrics: (1) the *success rate*, the percentage of instances where collision-free trajectories are found within 300 seconds time limit, (2) the *sum of travel times*, the sum of the travel times of all agents, and (3) the *runtime*, the computational time required to find collision-free trajectories, measured only for successfully solved instances.

Figure 4 shows our experimental results. With regard to the sum of travel times, Dual-CBS outperforms SSSP. Dual-CBS achieves a sum of travel times of 68.86s for five agents, while SSSP achieves only a sum of travel times of 131.02s. This gap widens as the number of agents grows. These differences stem from the different algorithmic approaches. Dual-CBS plans paths for all agents simultaneously, while SSSP plans for them sequentially, which forces agents to wait for other agents.

With regard to runtime, Dual-CBS needs only 0.30 to 3.61 seconds for smaller scenarios with 5 to 15 agents compared to 1.88 to 10.94 seconds for SSSP. Both algorithms need around 18 seconds for 20 agents. Dual-CBS needs 44.96 seconds compared to 27.63 seconds for SSSP.

With regard to success rate, both algorithms achieve perfect success rates of 100% for 5 to 20 agents. Dual-CBS achieves a success rate of 80% for 25 agents while SSSP continues to achieve near-perfect success rates.

Conclusion

In this paper, we proposed *Dual Conflict-Based Search* (Dual-CBS) for solving the MAMP problem. Dual-CBS achieves better solution quality than SSSP while avoiding the computational overhead of offline roadmap construction. Dual-CBS partitions the given configuration space into a grid and finds a set of grid-based paths by resolving conflicts with PCBS. It then finds a set of trajectories by sampling one configuration per grid cell and resolving collisions with SCBS, which uses swept area constraints to restrict sampling within the swept area of the trajectory of another agent. Empirically, Dual-CBS requires larger runtimes than SSSP to find collision-free trajectories for MAMP as the number of agents increases. However, Dual-CBS finds collision-free trajectories with a lower sum of travel times than SSSP. Our future work includes taking kinodynamic constraints into account when generating trajectories with SCBS.

Acknowledgments

The research at the University of California, Irvine and the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 2544613, 2434916, 2321786, 2112533, as well as gifts from Amazon Robotics and the Donald Bren Foundation.

References

Abdel-Malek, K.; Yang, J.; Blackmore, D.; and Joy, K. 2006. Swept volumes: Foundation, Perspectives, and Applications. *International Journal of Shape Modeling*, 87–127.

Cohen, L.; Uras, T.; Kumar, T. K. S.; and Koenig, S. 2019. Optimal and Bounded-Suboptimal Multi-Agent Motion Planning. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 44–51.

Dobson, A.; and Bekris, K. E. 2014. Sparse Roadmap Spanners for Asymptotically Near-Optimal Motion Planning. *International Journal of Robotics Research*, 18–47.

Henkel, C.; and Toussaint, M. 2020. Optimized Directed Roadmap Graph for Multi-Agent Path Finding using Stochastic Gradient Descent. In *Proceedings of the Annual ACM Symposium on Applied Computing*, 776–783.

Hönig, W.; Preiss, J. A.; Kumar, T. K. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory Planning for Quadrotor Swarms. In *IEEE Transactions on Robotics*, 856–869.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. 2011. Anytime Motion Planning using the RRT*. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1478–1483.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 566–580.

Kottinger, J.; Almagor, S.; and Lahijanian, M. 2022. Conflict-Based Search for Multi-Robot Motion Planning with Kinodynamic Constraints. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 13494–13499.

LaValle, S. M. 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *The Annual Research Report*, 1–4.

Okumura, K.; and Défago, X. 2023. Quick Multi-Robot Motion Planning by Combining Sampling and Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 252–261.

Reif, J. H. 1979. Complexity of the Mover's Problem and Generalizations. In *Proceedings of the Annual Symposium on Foundations of Computer Science (SFCS)*, 421–427.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 40–66.